DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-1

Algorithm: it is step by step procedure to solve a particular task or problem.

Or

It is a finite number of steps which can be used to solve a particular task or problem in step by step procedure.

Abu jafar mohammed ibn musa al khwarizmi in 780dc.

Notation of an algorithm:

Problem to be solved

Algorithm to perform a task divide and conquer/greedy method/dynamic programming/backtrackin

Input

----------------------→ Compute c c++ java ----------------------→ output correct wrong

Properties of an algorithm:

1.input:
2.output
3.definitess
4.effectiveness
5.finiteness

1.input: algorithm may take zero or more number of inputs

```
Void main()
{
printf("WELCOME");
}
```

----→ welcome

----→a

----→b

```
Void main()
{
Int a,b,c;
Printf("enter a,b values");
Scanf("%d%d",&a,&b);
C=a+b;
Printf("c=",&c);
}
```

---→c

2. Output: an algorithm should produce at least one output. If there is no output it is simply not an algorithm.

3. Definiteness: each and every statement should be clearly stated.
2+5=7 yes

10+2=12yes
10-5=5yes
10/0= infine wrong

4. effectiveness: each and every statement should effective. An algorithms does’t contain unnecessary statements.

```
Void main()
{
Int a,b,c;//necessary statement
Printf(“enter a,b values”);// may or maynot
Scanf(“%d%d”, &a,&b);//necessary statement
C=a+b;//necessary statement
Scanf(“%d”,&c);// un necessary statement
Printf(“c=”,&c);// necessary statement
}
```

5. Finiteness: an algorithm must be terminating at any particular point. If there is no terminating point simply it is not an algorithm.

Ex:

```
For(int i=0;i<=5;i++)
{

}
```

Output: 0 1 2 3 4

04-12-2021

Pseudo code for expressing algorithm

Pseudo code is an artificial and informal language which can be used to implement an algorithm.

Pseudo code is text based language which can be used to implement an algorithm.

Algorithm is basically sequence of instructions written in simple English language.

Algorithm can be represented in two ways

1.flow chart---------graphical representation of an algorithm
2.pseudo code -------it is text based representation of an algorithm by using some programming constraints.

Representation of an algorithms using pseudo code

1.each algorithm must starting with head and body

Algorithm algorithm-name(parameter1,parameter2..)

The body may represents with open brace and closed with closed braces.
{-------→ begin

}----→ending

Ex:

algorithm add(a,b,c)
Begin

Ending

2.every statement must ending with (;) delimeter

3.single line comments are written as '//' beginning of comment
4.identifers must starting with letters but not the digits
Ex: value-----yes
9value------wrong

5.if your assigning a value to the variable we can use assignment operator(:=)

Syntax:

Variable :=value; Ex: a:=10;

6.there are some operators are

Ex: <, <=, >, >=

7. The input and output statements can be declared as
In c language we can use printf() statement but where as in algorithm we can use write
Printf -------for output ----------algorithm---→write

Scanf-------for input= algorithm------→ read

8. the conditional statements such as if-then-else can be declared as

```
If(condition)
Begin
     Write("  a is large");
Ending

Else
Begin
      Write("b is large");
ending
```

9. for loop can be declared as

Program:

```
For(int a=0;a<=9;a++)
{
Printf("welcome');
}
```

Algorithm:

```
For variable:=value1 to value2 do
Begin

Write("welcome");

Ending
```

10. the while loop can be declared as

Syntax:

```
While(condition)
begin
```

```
    Statement1;
     Statement2;
Ending
```

11. the switch case can be declared as

```
Switch(value)
Begin
  Case1: statement;
              Break;
Case2:statement;
            Break;
Ending
```

12.functions can be declared as

```
Returntype function-name(parameters)
Begin

   Statements;

Ending
```

1.implement addition of two numbers algorithm using pseudo code

```
Algorithm addition(a,b,c)
Begin
      Write("enter a, b vales");
       Read("read a, b values");
       C:=adding a,b values;
       Write("display c values");
Ending
```

2.implementing whether the given number is even or odd algorithm using pseudo code(value%2==0)

```
Algorithm evenodd(a)
Begin
       Write("enter a value");
      Read("read a value"):

   If(condition)
   Begin
         Write("given number is even");
   Ending

  Else
  Begin
          Write("given number is odd");
  Ending

ending
```

06-12-2021

Approaches of an Algorithm

1.Priori analysis
2.Posterior analysis

1.Priori analysis: before executing the algorithm we will give behaviour or performance of an algorithm. It is not giving exact result. Less accurate.

2.Posterior analysis: after executing the algorithm we will measure the execution time or performance of an algorithm. It will gives exact results or accurate results.

Best case: which algorithm will takes less amount of time to complete the task or problem that is best algorithm. Which problem will takes less number of steps to complete that is best case.

Worst case: which algorithm will takes max amount of time to complete the task or problem that is worst algorithm. Which problem will takes max number of steps to complete that is called worst case.

Average case: which algorithm will takes average amount of time to complete the task or problem that is average algorithm. Which problem will takes average number of steps to complete that is called average case.

Ex: Linear Search (Sequential Search)

Array=

2	4	5	6	8	9	10	11	13	14

Problem: search the element 2 in an array using posterior analysis

Best case:
Sol:searching starting from zero index

A[0] -----→2 and our searching element is 2
Element 2 is in the a[0]. So we can solve the problem within a single step
Time complexity O(1)

Average case:
Problem: searching the element 8 is in the array or not

Sol:
Fisrt a[0] compare with 8 (2=8) not matching
A[1] compare with (4=8) not matching
A[2] compare with (5=8) not matching
A[3]compare with (5=8) not matching
A[4] compare with (8=8) both are matching

So the time complexity is O(n/2)

Worst case:

Problem. Searching the element 14 whether the element is in the array or not.

Sol:

A[0]---------→2 not matching
A[1]--------→4not matching
A[2]-------→5 not matching
A[3]---------6 not matching
A[4]-------→8 not matching
A[5]------→9 not matching
A[6]-------→10 not matching

A[7]------→11 not matching

A[8]------→13 not matching
A[9]------→14 matching means searching element is in the array index a[9]

Time complexity O(n)

08-12-2021

Performance of an algorithm

The efficiency of an algorithm can be decided by measuring the performance of an algorithm.

We can measure the performance of an algorithm by using two factors

1. Space complexity
2. Time Complexity

1. Space Complexity: The amount of memory taken by the algorithm to complete the task or problem that is called space complexity.

The amount of memory required by an algorithm during the execution of the algorithm.

Space complexity means space to store only data values but not the space to store the algorithm itself.

Ex: Int a,b   a=2bytes b=2byes inside the ram

Space complexity can be calculated by using the equation

S(p)=C+SP

Where S(P)=space of problem
C=constant variables or fixed variables---→means those are not dependent variables
Sp=instance variables or variable parts---→ means those are depends on some other factors.

Ex1:

Algorithm display()
Begin
    Write("welcome");
End

S(p)=C+SP

No variables are declared in this algorithm so c=0 and sp=0
S(p)=0 no memory is required for this algorithm

Ex2:

Algorithm add(a,b,c)
Begin
    Write("enter a,b values");
    Read("read a, b values");
    C=add a, b values;
    Write("display c value");
Ending

S(p)=C+SP
A variable occupy 1 memory space
B variable occupy 1 memory space
C variable occupy 1 memory space
There is no instance variables so SP=0

Space complexity S(p)= C+SP
=3+0
S(p)=3

Ex3:

```
Algorithm add(x,n)
Begin
          Sum:=0;
           For i:=1 to n do
                     Sum:=sum+x[i];
Ending
```

Sum 2 byes   i

1	1

N

1

X[i] requiry n memory space

Space complexity S(p)=C+SP
=3+n

Ex4:

```
 Algorithm display(i,n)
Begin
        For i:=1 to n do
        begin
       Write("welcome");
       end
End
```

Space complexity S(p)=C+SP

I occupy 1 memory space
N occupy 1 memory space
There is no instance variables sp=0
S(p)=2+0---→2 memory spaces

Ex5:

```
 Algorithm multiply(i,n,a,b,c)
Begin
       For i:=1 to n do
       Begin
         C:=a[i]+b[i];
      Ending
Ending
```

I occupy 1 memory space
N occupy 1 memory space
C variable occupy 1 memory space
A[i] occupy n memory space because it is depends up on n value
B[i] occupy n memory space because it is depends up on n value

Space complexity S(p)=3+2n memory spaces

2. Time complexity: the amount of time taken by an algorithm to complete the task or problem.

Time complexity is measured by using frequency count. Frequency count means how many number of times that the statements are executed.

Ex1:

s.no	**algorithm**	**Frequency count**
1	Algorithm display()	0
2	begin	0
3	Write("welcome")	1
4	end	0

Frequency count=0+0+1+0=1

Time complexity= O(1)

Ex2:

s.no	Algorithm	Frequency count
1	Algorithm display(i,n)	0
2	begin	0
3	For i:=1 to n do	N+1
4	begin	0
5	Write("welcome")	n
6	end	0
7	End	0

Frequency count=0+0+n+1+0+n+0+0
=2n+1

Time complexity=O(2n+1)

Note: when you want to calculate time complexity first we eliminate the constants. And we will take the upper polynomial time

Time complexity=O(n)

Ex4:

s.no	Algorithm	Frequency count
1	Algorithm add(a,b,c,i,j,n)	0
2	begin	0
3	For i:=1 to n do	N+1
4	Begin	0
5	For j:= 1 to n do	N(N+1)
6	Begin	0
7	C[i,j]:=a[i,j]+b[i,j]	N*n
8	end	0
9	end	0
10	end	0

Frequency count =0+0+n+1+0+n(n+1)+0+n2+0+0+0
=n+1+n2+n+n2+1
=2n2+2n+2

Time complexity =O(2n2+2n+1)

Note: when you want to calculate time complexity first we eliminate the constants. And we will take the upper polynomial time

Time complexity=O(n2+n)

=O(n2)

Ex:

s.no	Algorithm	Frequency count
1	Algorithm mul(a,b,c,i,j,n)	0
2	begin	0
3	For i:=1 to m do	M+1
4	Begin	0
5	For j:= 1 to n do	M(N+1)
6	Begin	0
7	C[i,j]:=a[i,j]*b[i,j]	M*n
8	end	0
9	end	0
10	end	0

Frequency count=0+0+m+1+0+m(n+1)+0+mn+0+0+0
=m+1+mn+m+mn
=2mn+2m+1

Time complexity=O(2mn+2m+1)

Note: when you want to calculate time complexity first we eliminate the constants. And we will take the upper polynomial time

Time complexity=O(mn+m)
=O(mn)

Analysis of linear search

```
int main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");------1
  scanf("%d", &n);---------------------------1
  printf("Enter %d integer(s)\n", n);---------1
  for (c = 0; c < n; c++)-------------------------n+1
    scanf("%d", &array[c]);-----------------------n
  printf("Enter a number to search\n");  -----------1
  scanf("%d", &search);       -----------------------1
  for (c = 0; c < n; c++)---------------------------n+1
  {
    if (array[c] == search)
    {
      printf("%d is present at location %d.\n", search, c+1);------n
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);-----------1
  return 0;
}
```

Analysis of linear search

space complexityS(P)= 3+n
search=1, n=1,c=1 sp=n

time complexity= 4n+8
=O(4n+8)
=O(n)

09-12-2021

Asymptotic Notations

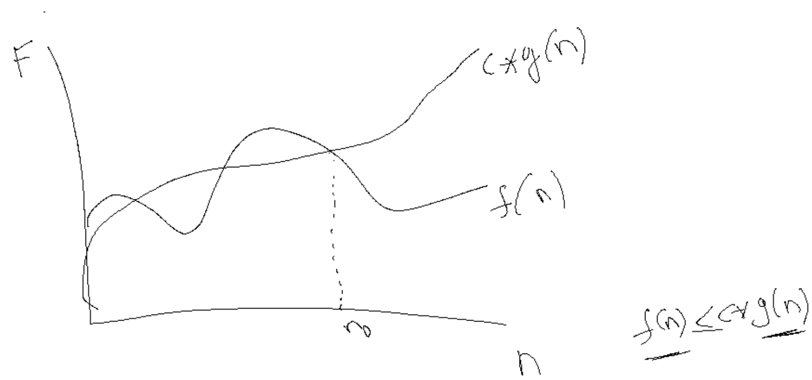
To choose the best algorithm we need to check efficiency of an algorithm. Efficiency of an algorithm is depends upon time complexity.

Asymptotic notation is a shorthand way to represent the time complexity of an algorithm. Using asymptotic notations we can give best worst and average.

1.Big oh Notation
2.Omega Notation
3.Theta Notation
4.Little on Notation
5.Little omega Notation

1.Big oh Notation: it is represented by "O". It is method of representing upper bound of an algorithm running time. Using big oh notation we can give longest amount time taken by the algorithm to complete.

Definition: let f(n) and g(n) are two non-negative functions. And there exists an integer n0 and constant c such that c>0 and for all integers n>n0, then **F(n)<=c*g(n).** Then f(n)=O(g(n))



EX: consider the function f(n)=2n+2 and g(n)=n2 we have to find constant c such that f(n)<=c*g(n)

Sol:
N=1 and c=1

F(n)<=c*g(n)

2n+2<=c*n2
2(1)+2<= 1*(1)2
4<=1------------------------→false
N=2 and c=1

2(2)+2<=1*(2)2
6<=4--------------→false

N=3 and c=1

2(3)+2<=1*(3)2
8<=9----------→true

N=4 and c=1

2(4)+2<=1*(4)2
10<=16-------→true

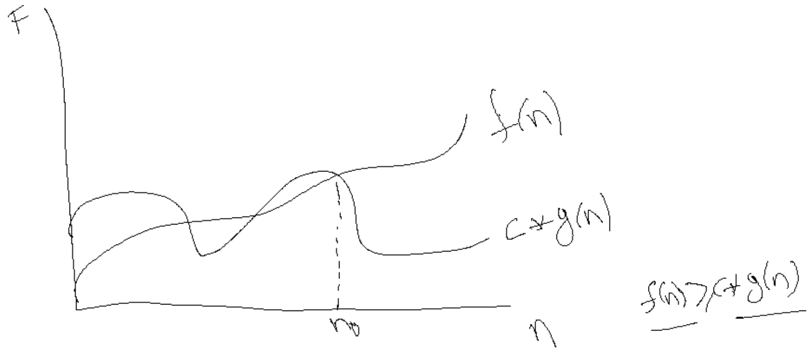
N=5 and c=1

2(5)+2<=1*(5)2
12<=25-------→true

Where n=3 and c=1 we are satisfying the Big oh notation
So time complexity f(n)=O(g(n))
=O(n2)

2. Omega Notation: it is denoted by"Ω". It is method of representing lower bound of an algorithm running time. Using the omega notation we can calculate shortest amount of time taken by the algorithm to complete.

Definiton: let f(n) and g(n) are two non-negative functions and there exists an integer and constant where c>0 and n>n0 then **f(n)>=c*g(n)**
Then f(n)= **Ω** (g(n)).



Example: let f(n)=3n+3 and g(n)=2n+5 and calculate c and n values and satisfy the omega notation**.**

Sol:

N=1 and c=1 then

F(n)>=c*g(n)

3n+3 >=c*2n+5

Substitute the n , c values

3(1)+3>=1*2(1)+5

6>=7-----→false

N=2 and c=1

3n+3>=c*2n+5

3(2)+3>=1*2(2)+5

9>=9----→true

N=3 and c=1

3n+3>=c*2n+5

3(3)+3>=1*2(3)+5

12>=11-----→true

N=4 and c=1

3(4)+3>=1*2(4)+5
16>=13----→true

Where c=1 and n=2 we can satisfy the condition f(n)>=c*g(n)
Then time complexity = Ω(g(n))
= Ω(2n+5)
= Ω(n)

Ex: let F(n)=2n+5 and g(n)=n and calculate c and n values such that f(n)>=c*g(n)

Sol:

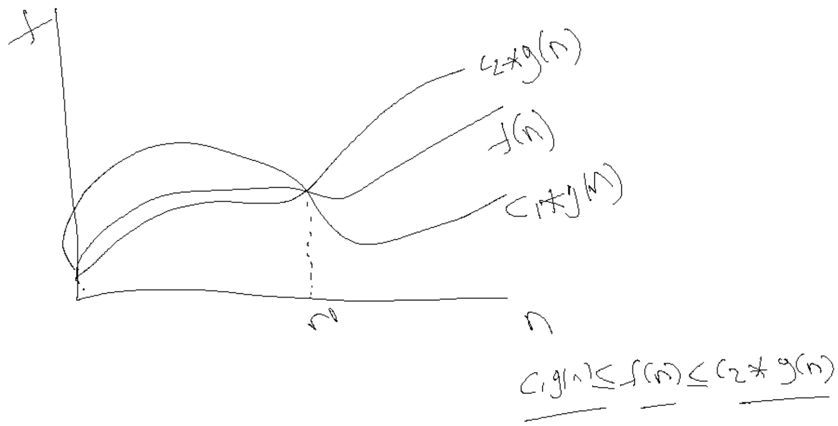
N=1 and c=1 then

2(1)+5>=1*1

7>=1----->true
Where n=1 and c=1 we are satisfying the condition f(n)>=c*g(n)
Time complexity = Ω(n)

3. Theta Notation: it is denoted by "φ". It is a method of representing average bound of an algorithm running time. Means we can calculate average amount of time taken by the algorithm to complete.

Definition: let there exists two non-negative function f(n) and g(n) and there exists constants c1, c2 and integer n, then c1<c2 and n>n0 such that c1*g(n)<=f(n)<=c2*g(n).



prob: f(n)=2n+8 g(n)=n then calculate c1,c2 and n values such that satisfy theta notation.

Sol: c1*g(n)<=f(n)<=c2*g(n)
C1<c2
C1=2 and c2=7 and n=1
C1*n<=2n+8<=c2*n
Substitute the values
2*1<=2(1)+8<=7(1)
2<=10<=7---------false

N=2

2*2<=2(2)+8<=7(2)
4<=12<=14-------true

N=3

2*3<=2(3)+8<=7(3)
6<=14<=21--------true

Where c1=2 and c2=7 and n>1 we can satisfy the theta notation now

Time complexity= φ(g(n))
= φ(n)

4.little oh notation:it is denoted by “o” let f(n) and g(n) are two non-negative function then

Lim f(n)\g(n)=0 then such that f(n)=o(g(n))
n->∞

5.little omega notation: it is denoted by” ɯ” let f(n) and g(n) are two non-negtive functions such that

Lim g(n)/f(n)=0 such that f(n)= ɯ(g(n))

n->∞

prob: determine f(n)=12n2+6n is O(n3)

sol: big oh notation condition is

f(n)<=c*g(n)
f(n)=12n2+6n
g(n)=n3

n=1 and c=2

12n2+6n<=c*n3
12(1)2+6(1)<=2*(1)3
18<=2----------false

N=2 and c=2

12(2)2+6(2)<=2*(2)3

48+12<=2*8
60<=16-----false

N=3

12(3)2+6(3)<=2*(3)3

108+18<=2*27
126<=54----false

N=4

12(4)2+6(4)<=2*(4)3
192+24<=2*64
216<=128-----false

N=5

12(5)2+6(5)<=2*(5)3

12*25+30<=2*125
330<=250

N=6

12(6)2+6(6)<=2*(6)3
12*36+36<=2*216

432+36<=432
468<=432

N=7

12(7)2+6(7)<=2*(7)3
12*49+42<=2*343

588+42<=686
630<=686---TRUE

Where c=2 and n=7 we are satisfy the big oh notation such that

Time complexity =O(g(n))
=O(n3)

or

N=3 and c=5

12(3)2+6(3)<=5*(3)3
126<=5*27
126<=

Prob: f(n)=4n2-64n+288 =Ω(n2)
Sol: omega notation f(n)>=c*g(n)
LHS f(n)=4n2-64n+288

N=1 c=5 then

F(n)=4(1)2-64(1)+288
=4-64+288
=228
RHS c*g(n)
=C*n2
=5*(1)2
=5

LHS>=RHS
F(n)>=c*g(n)
F(n)= Ω(g(n)) is proved

15-12-2021
Analysis of Insertion sort

Insertion sort works similar to the sorting of playing play cards. It is assumed that the first card is already in sorted order. We can select an unsorted card then compare with sorted card.
If the unsorted card is greater than the sorted card then we can place at right hand side.
If the unsorted card is less than the sorted card then we can place at left hand side.

Algortihm:

1.assume that the first element is in sorted order
2.select the next element and sorted it separately[compare with current element to its predecessor]
3.if the element is greater than the predecessor then we can we place at right hand side
4.if the element is less than the predecessor then we can place at left hand side(swapping condition)
5. repeat the steps 2 to 4 up to all the elements are placing in the sorted order.

3	6	2	1	8

------sorted--| ←-----unsorted sub array----------------------------→

Ex:

12	31	25	8	32	17

--sorted- | ←-----unsorted sub-array-----------------------------------→

Step-1: assume the first element in the array is sorted sub-array means the element is in sorted order.

Step-2: select next element from the unsorted sub-array and compare with elements in the sorted sub-array
Step-3: now 31 is greater than 12 that means 31 is placing at right hand side (31>12) means 12 is placing at correct position.

Now 12 and 31 is in sorted sub-array

12	31	25	8	32	17

---sorted sub-array--→|←----------------unsorted sub-array-----------→

Step-4: now we can select the next element from the unsorted sub-array.
And compare with all the elements in the sorted sub-array.
Now 25 is less than 31(25<31) then we can placed at left hand side.
Swap 25 with 31

12	31	25	8	32	17

After swapping

12	25	31	8	32	17

Now again 25 is comparing with element 12 now 25 is greater than 12 (25>12)means 25 is placing at right hand side. Now the sorted array is

12	25	31	8	32	17

←------------sorted sub-array--→ |←----unsorted sub-array------→

Step-5: again select the next element from the unsorted sub-array. Now 8 is comparing with all the elements in the sorted sub-array.

Now 8 is compare with element 31, now 8 is less than 31(8<31) then we can placed at left hand side. Apply swapping now 8 is swapping with element 31

12	25	31	8	32	17

After swapping the array is

12	25	8	31	32	17

Step-6: Now again 8 is comparing with element 25. Now 8<25 means we can placed at left hand side. We can apply swapping condition

Now before swapping the array is

12	25	8	31	32	17

After swapping the array is

12	8	25	31	32	17

Step-7: Now again element 8 is comparing with element 12 which is less than the element 12. (8<12) means 8 is placing at left hand side. We can swapping condition. 8 is swapping with element 12

12	8	25	31	32	17

After swapping the array is

8	12	25	31	32	17

←--------sorted sub-array-----------------------→ |←unsorted sub-array→

Step-8: we can select next element from unsorted sub-array. And comparing with all the elements in the sorted sub-array.

Now 32 is comparing with its predecessor 31. Now 32 is greater than 31 (32>31) now we can placed at right hand side. That means 31 is placing at right position. Again 32 is comparing with 25(32>25) we can placed at right hand side only. 32 is comparing with element 12 (32>12) we can placed at right hand side only. Again 32 is comparing with element 8(32>8) we can placed at right hand side only.

After than

8	12	25	31	32	17

←-------------------sorted sub-array-------------------------→ |<un sorted>

Step-9: now again we can select the next element from unsorted sub-array. And compare with all the elements in the sorted sub-array. Now 17 is comparing with its predecessor element 32. Now 17 is less than 32(17<32) we can placed at left hand side. Apply swapping condition now 17 is swapping with element 32

8	12	25	31	32	17

After swapping

8	12	25	31	17	32

Step-10: now element 17 is comparing with element 31. 17 is less than 31(17<31) now we can placed at left hand side. Swapping condition is apply. Now 17 is swapping with element 31

8	12	25	31	17	32

After swapping

8	12	25	17	31	32

Step-11: now again the element 17 is comparing with the element 25. Now 17 is less than 25(17<25). Now 17 is swapping with element 25

8	12	25	17	31	32

After swapping

8	12	17	25	31	32

Step-12: now again element 17 is comparing with the element 12. Now 17 is greater than 12. Now we can placed at right hand side.

8	12	17	25	31	32

Now all the elements in the array are placing in sorted order

Algorithm:

```
Algorithm insertion(i,n,a[])
Begin
   For i:=1 to n
      Temp:=a[i]
       J:=i-1
     While(j>=0&&a[j]>temp)
     Begin
       A[j+1]=a[j]
J=j-1;
```

End while
A[j+1]=temp
End for

End

Algorithm:

```
Algorithm insertion(i,j,n,a[])
{
For(i=1;i<n;i++)
{
Temp=a[i];
J=i-1;
While(j>=0&&a[j]>temp)-----compare whether it is greater or less than
{
A[j+1]=a[j]
J=j-1
}
A[j+1]=temp--------------storing statement
}-------ending for loop
}
```

Ex:

5	4	10	1	6	2

Sorted | ←------unsorted sub-array ------------------------------→

Sol:

Step-1: i=1

For(i=1;1<6;i++) ----------true

Temp=a[i]=a[1]
Temp=4

J=i-1
J=1-1=0
While(j>=0&&a[j]>temp)
While(0>=0&&a[0]>temp)
While(0>=0&&5>4)-----------true
{
A[j+1]=a[j]➔a[0+1]=a[0]➔a[1]=a[0]
A[1]=a[0] that means a[0]is swapping with a[1]

4	5	10	1	6	2

J=j-1➔0-1
J=-1
}

Again while loop is executed

While(j>=0&&a[j]>temp)
While(-1>=0&&a[-1]>temp) -------false

Out of while loop is executed

A[j+1]=temp
A[-1+1]=temp
A[0]=temp
That means element 4 is placed at a[0]

4	5	10	1	6	2

---sorted--------------→ | < ---- unsorted sub-array-----------------------→

Step-2: increment i value i=i++=i+1=1+1=2

For(i=2;2<6;i++)
{
Temp=a[i]
Temp=a[2]
Temp=10
J=i-1
J=2-1
J=1
While(j>=0&&a[j]>temp)
While(1>=0&&a[1[>temp)
While(1>=0&&5>10)------------false

If while loop is false then out if while loop is executed.

A[j+1]=temp
A[1+1]=10
A[2]=10

Again i value is incremented

Step-3: i=i+1=2+1=3

4	5	10	1	6	2

For(i=3;3<6;i++)
{
Temp=a[i]
Temp=a[3]
Temp=1

J=i-1
J=3-1
J=2

While(j>=0&&a[j]>temp)
While(2>=0&&a[2]>1)
While(2>=0&&10>1)-------------------ture
{
A[j+1]=a[j]
A[2+1]=a[2]
A[3]=a[2] means a[2] is swapping with a[3]

4	5	1	10	6	2

J=j-1
J=2-1
J=1
}

Again same while loop is executed

While(1>=0&&a[j]<temp)
While(1>=0&&a[1]>1)
While(1>=0&&5>1)------------true
{
A[j+1]=a[j]
A[1+1]=a[1]
A[2]=a[1] means a[1]is swpping with a[2]

4	1	5	10	6	2

J=j-1
J=1-1=0
J=0
}
Again same while loop is executed

While(j>=0&&a[j]>temp)
While(0>=0&&a[0]>temp)

While(0>=0&&4>1)--------------true
{
A[j+1]=a[j]
A[0+1=a[0]
A[1]=a[0] means =a[0] is swpping with a[1]

1	4	5	10	6	2

J=j-1
J=0-1
J=-1
}

Again while loop[
While(j>=0&&a[j]>temp)
While(-1>=0&&a[-1]>1)----false

Out of while loop is executed

A[j+1]=temp
A[-1+1]=temp
A[0]=temp -----means 1 is stored in a[0]

Step-3; i is incremented i=i+1

1	4	5	10	6	2

I=3+1=4
For (i=4;4<6;i++)
{
Temp=a[i]=a[4]=6
Temp=6
J=i-1
J=4-1=3
While(j>=0&&a[j]>temp)

While(3>=0&&a[3]>6)
While(3>=0&&10>6)-------true
{
A[j+1]=a[j]
A[3+1]=a[3]
A[4]=a[3] swap a[3] with a[4]

1	4	5	6	10	2

J=j-1=3-1=2
}

Again while loop is executed
While(j>=0&&a[j]>temp)
While(2>=0&&a[2]>6)
While(2>=0&&5>6)------------false

Out of while loop is executed

A[j+1]=temp
A[2+1]=temp
A[3]=temp
A[3]=6

Step-4: i value is incremented

1	4	5	6	10	2

I=i+1=4+1=5
For(i=5;5<6;i++)-----true
{
Temp=a[i]=a[5]=2
J=i-1=5-1=4
While(4>=0&&a[4]>2)
While(4>=0&&10>2)---------true

{
A[j+1]=a[j]
A[4+1]=a[4]
A[5]=a[4]

1	4	5	6	2	10

J=j-1
J=4-1
J=3
}
Again while loop executed

While(3>=0&&a[3]>temp)
While(3>=0&&6>2)-----true
{
A[j+1]=a[j]
A[4]=a[3]

1	4	5	2	6	10

J=j-1
J=3-1
J=2
}

Again while loop executed
While(2>=0&&a[2]>temp)
While(2>=0&&5>2)---------------true
{
A[j+1]=a[j]
A[2+1]=a[2]
A[3]=a[2] swap a[2] with a[3]

1	4	2	5	6	10

J=j-1

J=2-1
J=1
}

Again while loop is executed

While (j>0&&a[j]>temp)
While (1>=0&&a[1]>2)
While (1>=0&&4>2) ---------true
{
A [1+1] =a [1]
A [2] =a [1] swap a [1] with a [2]

1	2	4	5	6	10

J=j-1
J=1-1=0
}
Again while loop is executed

While (j>=0&&a[j]>temp)
While (0>=0&&a[0]>2)
While (0>=0&&1>2) ----false
Means 2 is placed in correct position
Out of for loop is executed

A[j+1]=temp
A[0+1]=temp
A[1]=temp
Element 2 is stored in a[1]

Finally i values is incremented

Step-5: i=i+1=5+1=6
For(i=6;6<6;i++)---------false

Finally all the elements are in sorted order ascending order

1	2	4	5	6	10