Instruction set of the processor

- ***1 step for execution:***
  - Arithmetic and logical operation: A=B+C…
  - Jumps and conditions: go to if a>b then y
  - Pointer instruction: B=*C, *C=B
  - 1 D array operation :A[i]=B or B= C[i]

Complex algorithmic instruction

A= B+C*D-F

A[i]=B[i]+C[i] {x=B[i], y=C[i], z=x+y, A[i]=z}

Example

- For i=1 to n
- C[i]= A[i]+B[i]
- End for
- How many steps are required to execute?
- What will be running time?

Example

- i=1 1 times
- If i>n go to last n+1 times

- X=A[i]
- Y=B[i] n times
- Z=X+y
- C[i]=Z

- i=i+1
- Go to … n times

Running time= b*n+ 2n+n+1+1 =bn+3n+2, =2+n(b+3)

Analysis of Algorithm

Matrix Multiplication

- Input: M and N with n*n
- Output: O with n*n
- O= M*N
- Oij= $\sum_k mik * nkj$
- For i=1 to n

  for j= 1 to n

  c[i,j]=0

Analysis of Algorithm

- For i=1 to n

  for j= 1 to n

  O[i,j]=0

  for k= 1 to n

  O[i,j]= O[i,j]+M[i,k]*N[k,j]

  End

  End

  End

Analysis of Algorithm

Body: b steps

Iteration: n

Time taken :2+n(b+3)

Accessing of 2 dimensional array: 4 steps

Analysis of Algorithm

```
for(i=0; i < N; i++)
{
    statement;
}
```

Complexity?

Analysis of Algorithm

```
for(i=0; i < N; i++)
{
    for(j=0; j < N;j++)
    {
    statement;
    }
}
```

Complexity?

Analysis of Algorithm

```
int a = 0;

for (i = 0; i < N; i++) {

    for (j = N; j > i; j--) {

        a = a + i + j;

    }

}
```

Complexity?

Analysis of Algorithm

```
while(low <= high)
{
    mid = (low + high) / 2;
    if (target < list[mid])
        high = mid - 1;
    else if (target > list[mid])
        low = mid + 1;
    else break;
}
```

Complexity?

Analysis of Algorithm

```
void sort(int list[], int left, int right)
{
    int pivot = partition(list, left, right);
     sort(list, left, pivot - 1);
     sort(list, pivot + 1, right);
}
```

Complexity?

Analysis of Algorithm

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```
Complexity?

Solution

- If you notice, j keeps doubling till it is less than or equal to n. Number of times, we can double a number till it is less than n would be log(n). Let's take the examples here.
  for n = 16, j = 2, 4, 8, 16
  for n = 32, j = 2, 4, 8, 16, 32
  So, j would run for O(log n) steps.
  i runs for n/2 steps.
  So, total steps = O(n/ 2 * log (n)) = **O(n*logn)**

- $f(n) = 7n + 8$ and $g(n) = n$.
- Is $f(n) \in O(g(n))$?

- For $7n + 8 \in O(n)$, we have to find c and n0 such that $7n + 8 \leq c \cdot n, \forall n \geq n0$. By inspection, it's clear that c must be larger than 7.
- Let c = 8. Now we need a suitable n0. In this case, f(8) = 8 .g(8).
- Because the definition of O() requires that $f(n) \leq c \cdot g(n)$, we can select n0 = 8, or any integer above 8 – they will all work.
- We have identified values for the constants c and n0 such that $7n + 8$ is $\leq c \cdot n$ for every $n \geq n0$, so we can say that $7n + 8$ is $O(n)$.