Q. 13 Suppose x is array of positive integers of size n, and y is another array of positive integers of size m. These two array represents the n-digits and m-digits long integers. Write a program to find the multiplication of x and y.

Algorithm MULTIPLY(a,b):

Input: 2 Numbers where digits of a >= b.
Output: Multiplication result of 2 numbers.

```
IF(a<10 or b<10) THEN DO
   RETURN (a*b)
END IF
set lenA = LENGTH(a)
set lenB = LENGTH(b)
set aUHalf = DIVIDE(0,lenA/2)
set bUHalf = DIVIDE(0,lenB/2)
set aLHalf = DIVIDE(lenA/2,lenA)
set bLHalf = DIVIDE(lenB/2,lenB)

set p0 = MULTIPLY(aUHalf ,bUHalf)
set p1 = MULTIPLY(aLHalf,bLHalf)
set p2 = MULTIPLY(aLHalf+aUHalf ,bLHalf+bUHalf) - p0 - p1
RETURN (p0*(POWER(10,lenA))) + (p2*POWER(10,lenA/2)) + p1
```

Algorithm Analysis:

$T(n) = 3*T(n/2) + c*\log(n) + c1$

on solving..

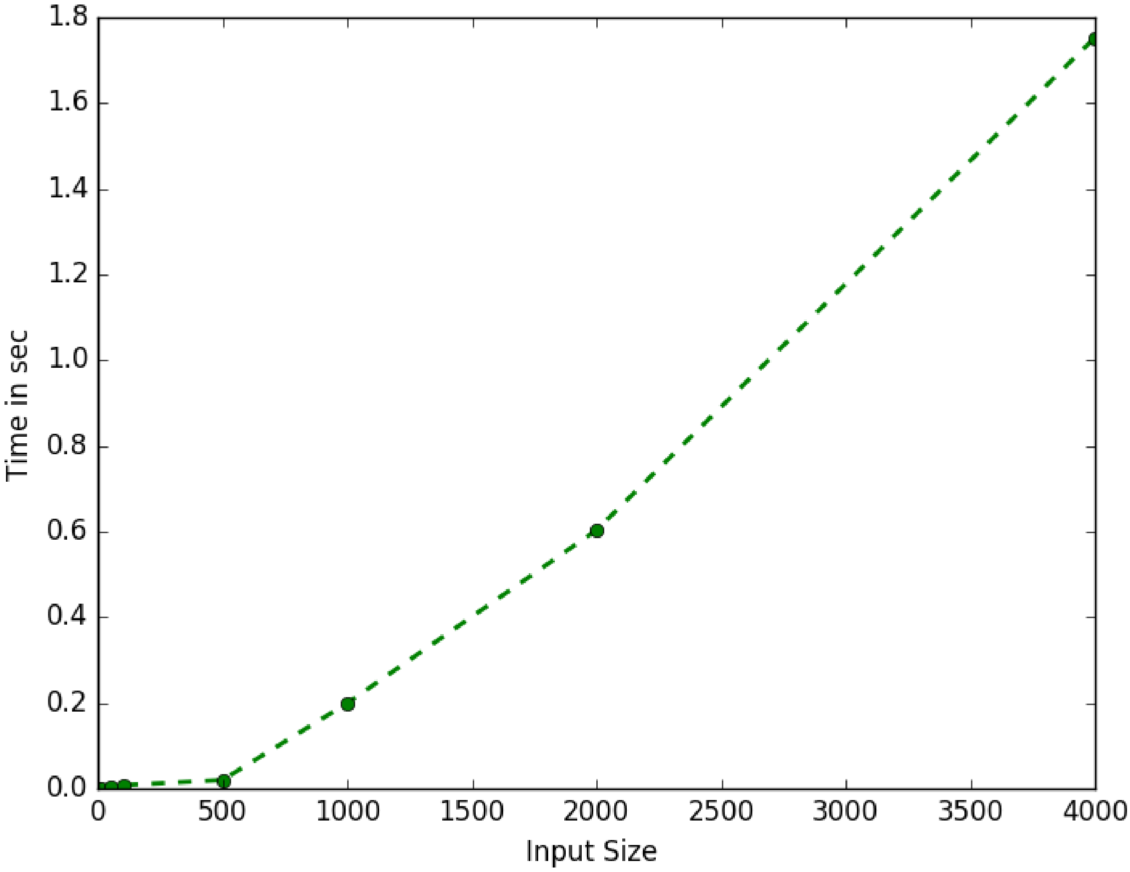
$T(n) = O(n^{\log_2 3})$

Input/Output:

Input is n = Number of digit size
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
1	0.0
10	0.0
50	0.004
100	0.008
500	0.02
1000	0.199
2000	0.603
4000	**1.753**

Complexity Graph:



Conclusion :
By using devide and conquer technique the algorithm reduce the complexity from n^2 to n^(log3) but still as it can be confirm by graph the algorithm takes polynomial time.

Q. 15 Consider the modified binary search algorithm so that it splits the input not into two sets of almost-equal sizes, but into three sets of sizes approximately one-third. Write down the recurrence for this ternary search algorithm and find the asymptotic complexity of this algorithm. Also write a program and verified it.

Algorithm TERNARYSEARCH(a,x,l,r):

Input: Numbers to search for and list a.
Output: Found or Not found.

```
IF(l<=r) THEN DO
     set ot = l + (r-l)/3;
     set tt = l + 2*(r-l)/3;
     IF(a[ot]==x) THEN DO
        RETURN TRUE;
     END IF;
     ELSE IF (a[ot]>x) THEN DO
        TERNARYSEARCH(a,x,l,ot-1);
     END IF;
     ELSE IF (a[tt]==x) THEN DO
        RETURN TRUE
     END IF;
     ELSE IF (a[tt]>x) THEN DO
        TERNARYSEARCH(a,x,ot+1,tt-1);
     END IF;
     ELSE DO
        TERNARYSEARCH(a,x,tt+1,r);
     END IF;
END IF;
ELSE DO
       RETURN FALSE;
END IF
END
```

Algorithm Analysis:

T(n) = T(n/3) + 2

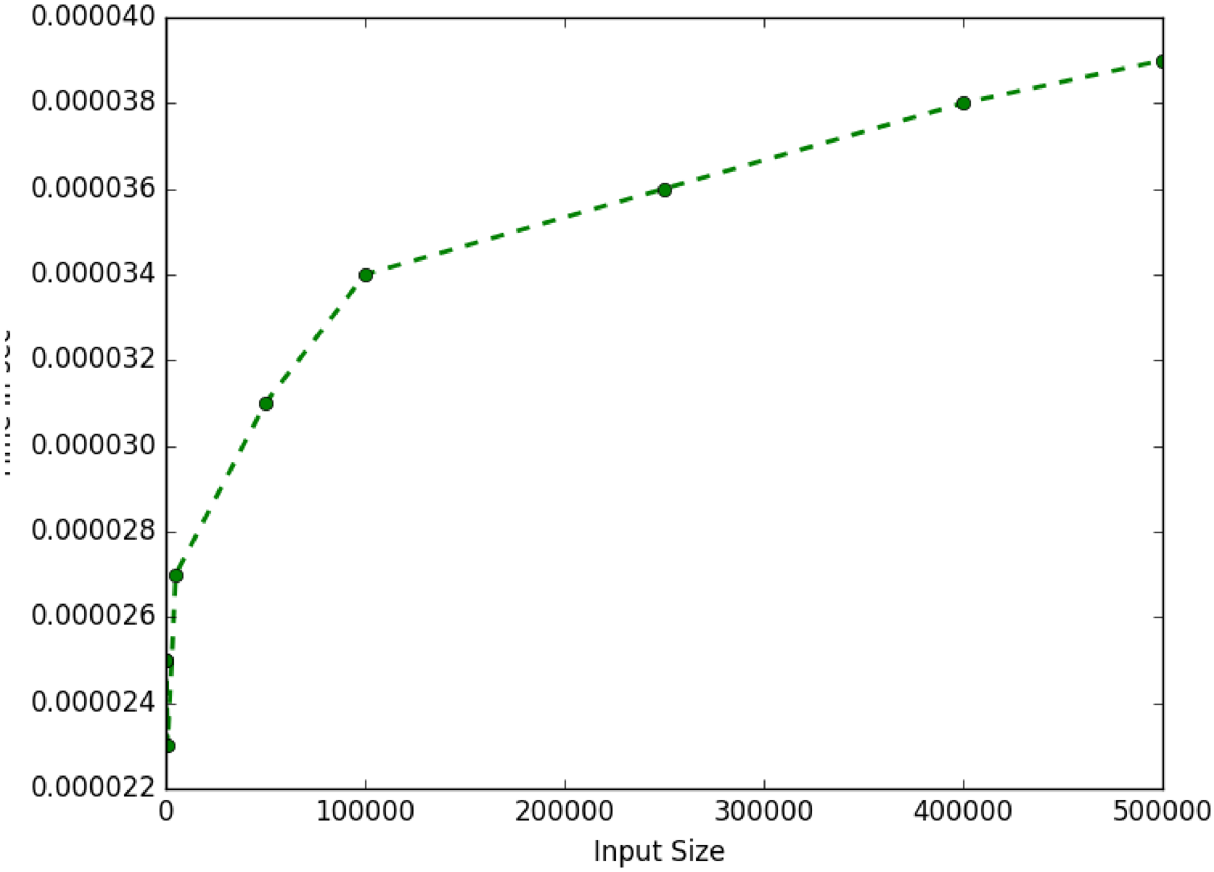
on solving..
T(n) = O(logn)

Input/Output:

Input is n = Size of list.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
1	0.000023
100	0.000025
5000	0.000027
50000	0.000031
100000	0.000034
250000	0.000036
400000	0.000038
500000	**0.000039**

Complexity Graph:



Conclusion :
IF we divide list into 3 equal halves the time complexity remains in logn time but comparision time for spliting list into 3 half, make it practically slower then binary search.

Q. 16 Consider another variation of the binary search algorithm so that it splits the input not only into two sets of almost equal sizes, but into two sets of sizes approximately one-third and two-thirds. Write down the recurrence for this search algorithm and find the asymptotic complexity of this algorithm. Also write a program and verified it.

Algorithm HALFTERNARYSEARCH(arr,x,l,r):

Input: Numbers to search for and list a.
Output: Found or Not found.

```
IF (r >= l) THEN DO
    set mid = l + (r - l)/3;
    if (arr[mid] == x) THEN DO
        RETURN mid;
    END IF
    IF (arr[mid] > x) THEN DO
        RETURN HALFTERNARYSEARCH(arr, l, mid-1, x);
    END IF
    RETURN  HALFTERNARYSEARCH(arr, mid+1, r, x);
END IF
RETURN FALSE;
```

Algorithm Analysis:

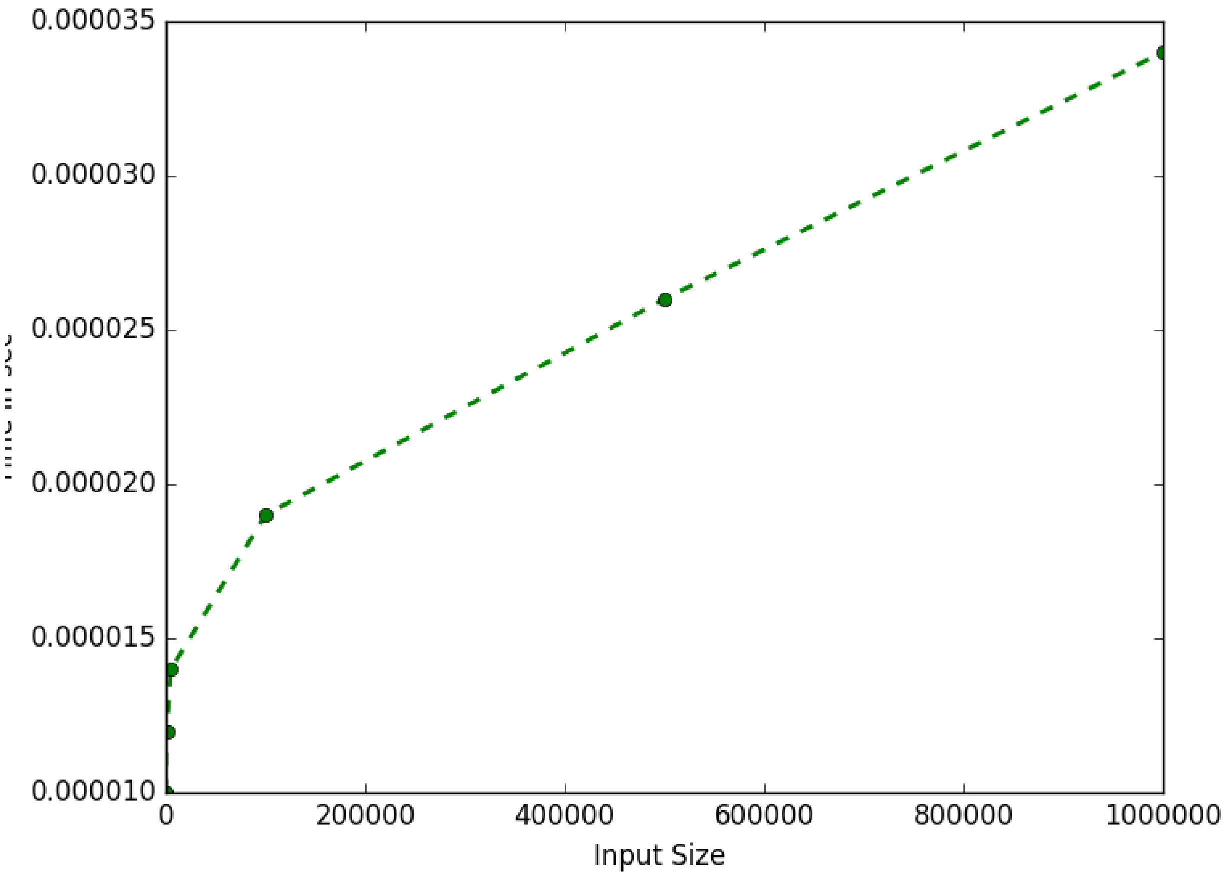
$T(n) = T(n/3) + 1$ if $x < a[n/3]$
$= T(2*n/3) + 1$ else
on solving..
$T(n) = O(\log n)$

Input/Output:

Input is n = Size of list.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
1	0.000010
10	0.000010
100	0.000012
1000	0.000012
5000	0.000014
100000	0.000019
500000	0.000026
1000000	**0.000034**

Complexity Graph:



Conclusion :
IF we divide list into 2 unequal halves the time complexity (logn) changes on the basis of where should be the element present in the list if it is in the initial one third part of list complexity is less than that of binary search and if in worst case it is at remaining 2/3rd part of list complexity more than binary search and for average case complexity should be logn but practically more than binary search.

Q. 17 Write a program and also analyze a divide and conquer MAXMIN algorithm that find minimum and maximum of given list of n integers.

Algorithm MAXMIN(arr,l,r,m,mi):

Input: Numbers List.
Output: Max and Min numbers in list.

```
set max=*m;
set min = *mi;
IF (l==r) THEN DO
  max=min=a[l];
END IF
ELSE IF (l==r-1) THEN DO
  if(a[l]>a[r])  THEN DO
    max=a[l];
    min=a[r];
  END IF ELSE DO
    min=a[l];
    max=a[r];
  END IF
END IF
ELSE DO
  set mid = l + (r-l)/2;
  MAXMIN(a,l,mid,&max,&min);
  MAXMIN(a,mid+1,r,&max1,&min1);
  IF (min1<min) THEN DO
     set min=min1;
  END IF
  IF (max1>max) THEN DO
    max=max1;
  END IF
 END IF
set *m=max;
set *mi=min;
```

Algorithm Analysis:

T(n) = 2*T(n/2) + C

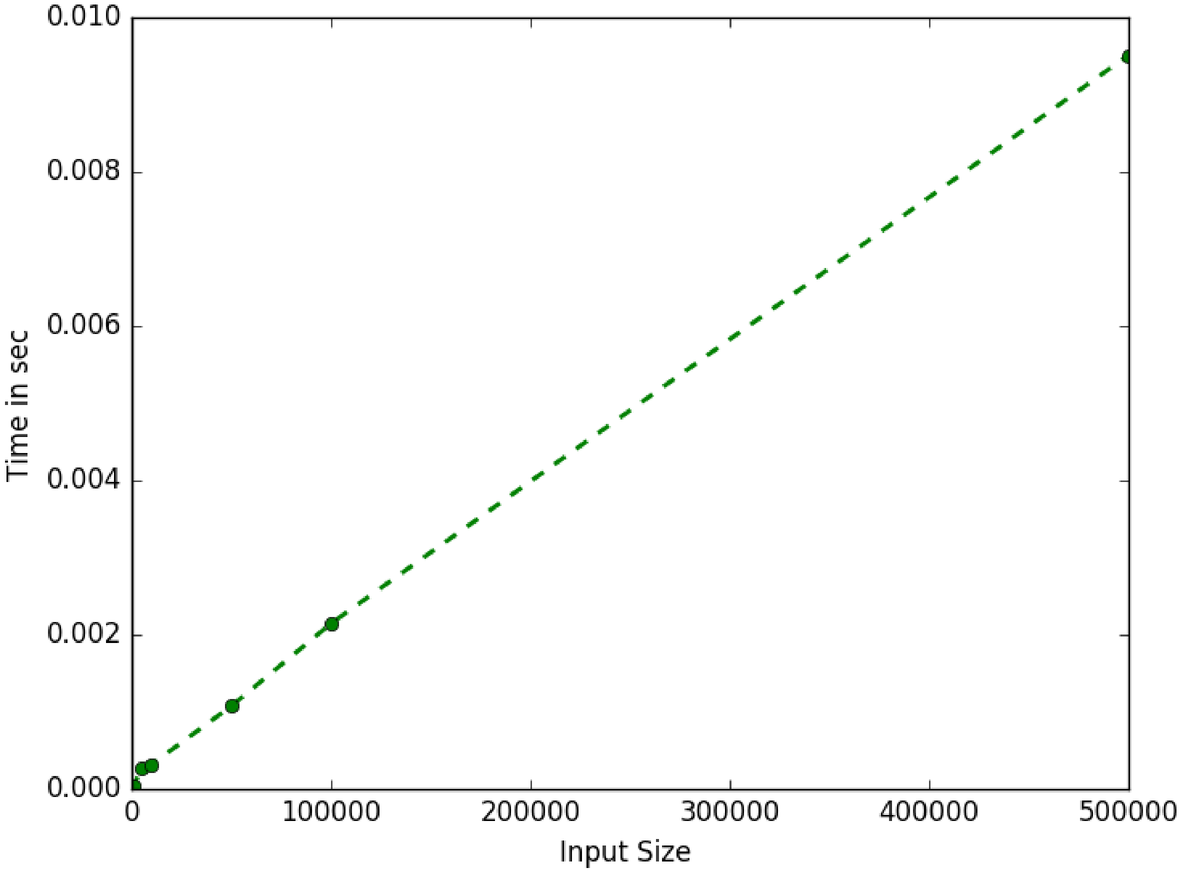
on solving..
T(n) = O(n)

Input/Output:

Input is n = Size of list.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
10	0.000003
100	0.000007
1000	0.000050
5000	0.000268
10000	0.000321
50000	0.001076
100000	0.002146
500000	**0.009510**

Complexity Graph:



Conclusion :
As expected graph show the nature of the algorithm as to be linear same the the theoratical complextiy O(n).

Q. 18 A set of points in the plane, {p1 = (x_1; y_1); p2 = (x_2; y_2); : : : ; p_n = (x_n; y_n)} are given, write a program to find the closest pair of points: that is, the pair pi != pj for which the distance between p_i and p_j , that is,$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$; is minimized.

Algorithm CLOSEST(a,l,r):

Input: X cordinate wise sorted List.
Output: Pair of point which are closest with their distance.

```
set x = [0,0]
set y = [0,0]
IF (r-l) <=3 THEN DO
   RETURN minDistance(a,l,r)
 set mid = l+(r-l)/2
 set midX = a[mid]
 set dMinL,x1,y1 = CLOSEST(a,l,mid)
 set dMinR,x2,y2 = CLOSEST(a,mid+1,r)
 set dmin = min(dMinL,dMinR)
 IF (dmin==dMinL) THEN DO
   set x,y = x1,y1
 ELSE
   set x,y = x2,y2
 middle = Φ
 FOR i L to r DO:
   IF abs(a[i][0]-midX[0]) < dmin THEN DO:
     middle = middle U {(a[i])}
 END LOOP
 set midMin,x,y = minimumToMid(middle,0,len(middle),dmin,x,y)
 IF (dmin < midMin) THEN DO:
   RETURN dmin,x,y
 ELSE DO:
   RETURN midMin,x,y
```

Algorithm Analysis:

T(n) = 2*T(n/2) + θ(n)

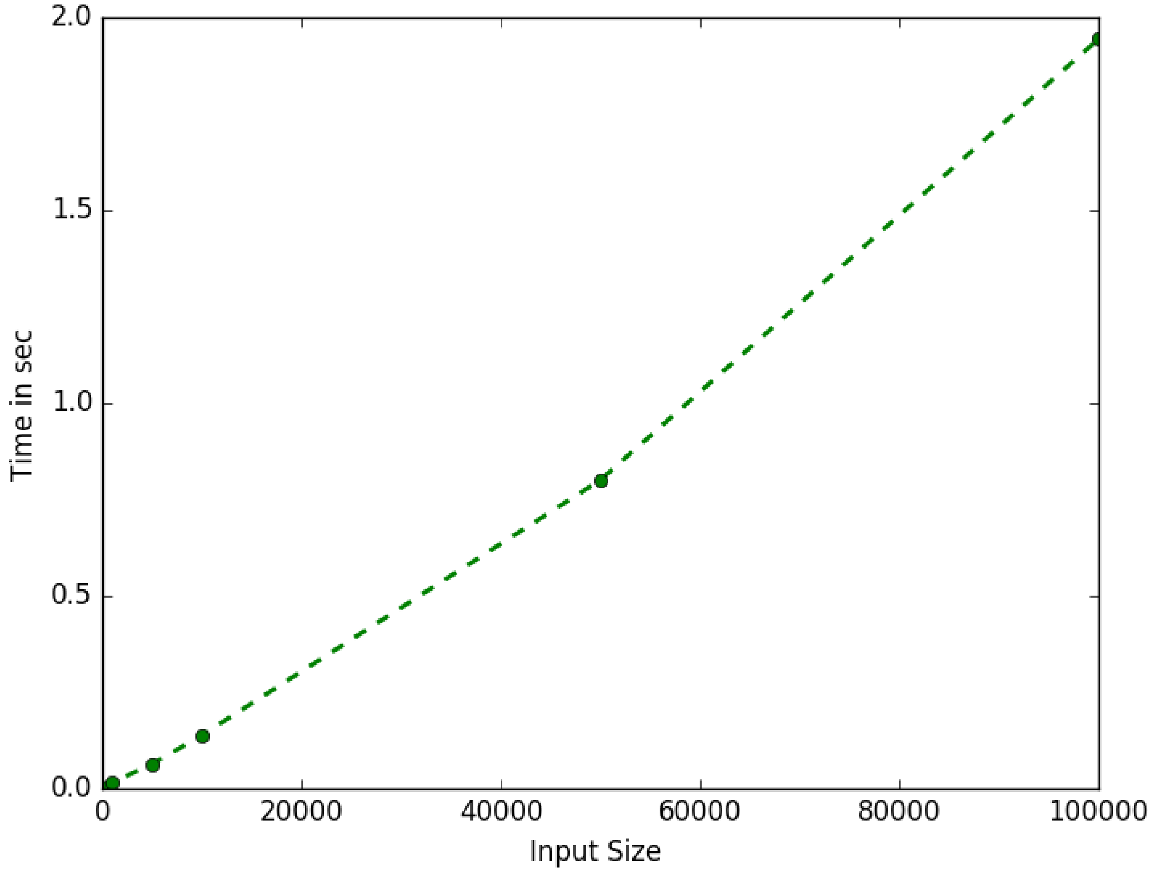
on solving..
T(n) = O(nlogn)

Input/Output:

Input is n = number of pair of points.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
10	0.00023
100	0.00088
500	0.00595
1000	0.01601
5000	0.06343
10000	0.13813
50000	0.8001
100000	**1.94391**

Complexity Graph:



Conclusion :
Complexity as per theoratical analysis should be nlogn but as we can see from graph it is slightly linear-polynomial type, this is may be due to the fact that computation for calculating distance from middle is not considered for theoratical analysis.

Q. 19 Suppose S is array of positive integers of size n is given. Write a program to find the maximum jump from an earlier index(i) to a later index(j), where i ≤ j. For example, if the array is [40; 20; 0; 0; 0; 1; 3; 3; 0; 0; 9; 21], then the maximum jump is 21, which happens between the index at 2 and index at 11. More formally, the problem is to compute: Max{ (Sj – Si) 0 ≤ i ≤ j ≤ |S| }.

Algorithm MAXJUMP(a,l,r,m,mi):

Input: List of numbers.
Output: Maximum jump occur in term of difference going from left to right.

```
 set max = *m;
 set min = *mi;
 set diff=0;
 IF(l==r) THEN DO:
    set max=min=a[l];
    set diff=0;
ELSE IF (l == (r-1)) THEN DO:
    set diff = a[r]-a[l];
    IF (a[l]>a[r]) THEN DO:
       set max=a[l];
       setmin=a[r];
    ELSE DO:
       set min  = a[l];
       set max = a[r];
   ELSE DO:
    set max1,min1=0;
    set mid = l + (r-l)/2;
    set diff1=diff2=diff3=0;
    diff1=MAXJUMP(a,l,mid,&max,&min);
    diff2=MAXJUMP(a,mid+1,r,&max1,&min1);
    set diff3=max1-min;
    IF (diff1>diff2) THEN DO:
       IF(diff1>diff3) THEN DO:
          set diff=diff1;
       ELSE
          set diff=diff3;
          set max=max1;
   ELSE DO:
       IF (diff2>diff3) THEN DO:
          set diff=diff2;
          set max=max1;
          set min=min1;
        ELSE DO:
          set diff=diff3;
          set max=max1;
 set *m = max;
 set *mi = min;
 RETURN diff;
```

Algorithm Analysis:

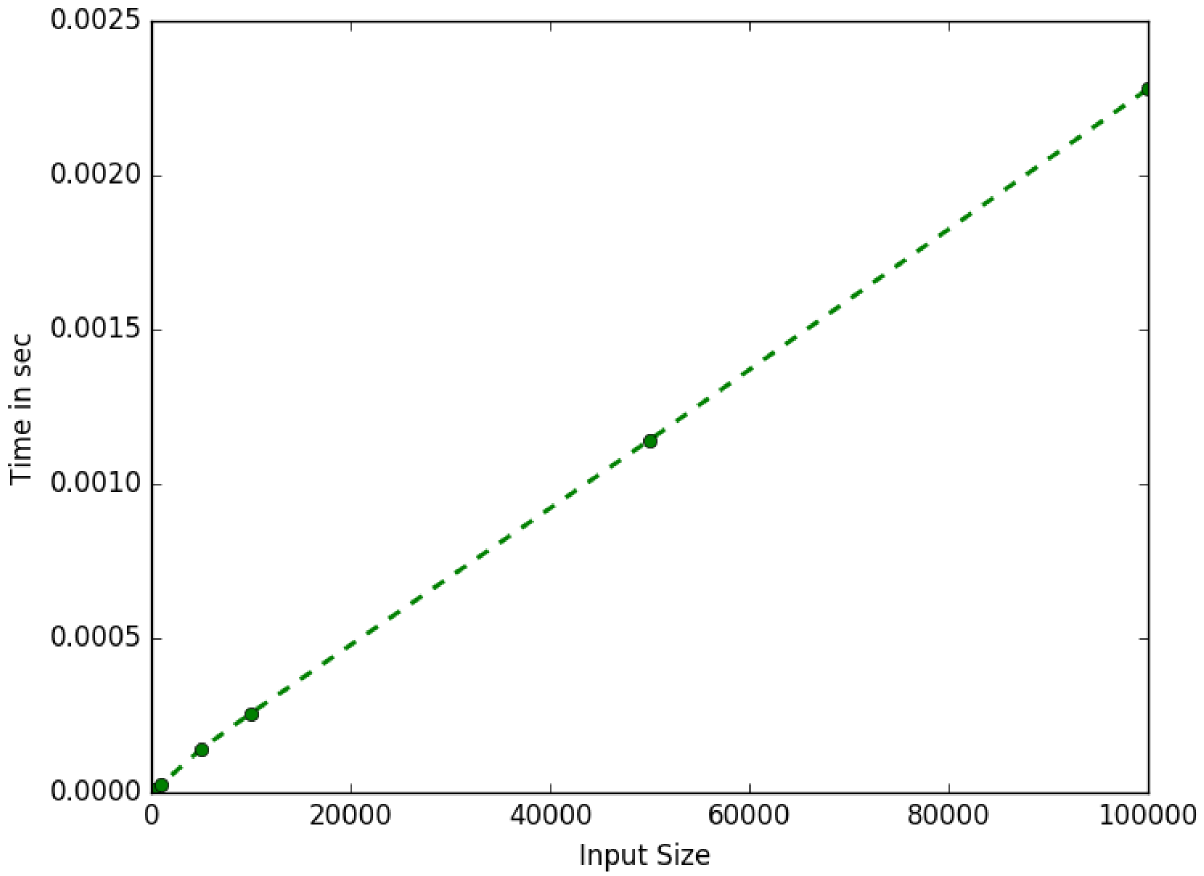
T(n) = 2*T(n/2) + C

on solving..
T(n) = O(n)

Input/Output:
Input is n = Size of list.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	**Time (in Sec)**
10	0.00023
100	0.00088
500	0.00595
1000	0.01601
5000	0.06343
10000	0.13813
50000	0.8001
100000	**1.94391**

Complexity Graph:



Conclusion :
As per algorithm it should be theoratically should be of complexity of linear time and as per graph it can verify that implemented program of above algorithm is executing in linear time.

Q. 20 Write a program to find the approximate nth Fibonacci number in O(log n) time.

Algorithm FIBONACCI(n):

Input: N for nth fibonacci.
Output: Nth fibonacci number.

```
POWER(x,y):
   IF (y==0) THEN DO:
     return 1;
  ELSE IF (y%2==0) THEN DO:
     set p = POWER(x,y/2);
     RETRUN p*p;
  ELSE DO:
     set p = POWER(x,y/2);
     RETURN x*p*p;

FIBONACCI(n):
       set root5 = √5;
       set phi = (1+root5)/2;
       set omega = (1-root5)/2;
       set fibo = (POWER(phi,n)-POWER(omega,n))/root5;
       RETURN F;
```

Algorithm Analysis:

T(n) = c*logn + c1 ...... logn due to power function.

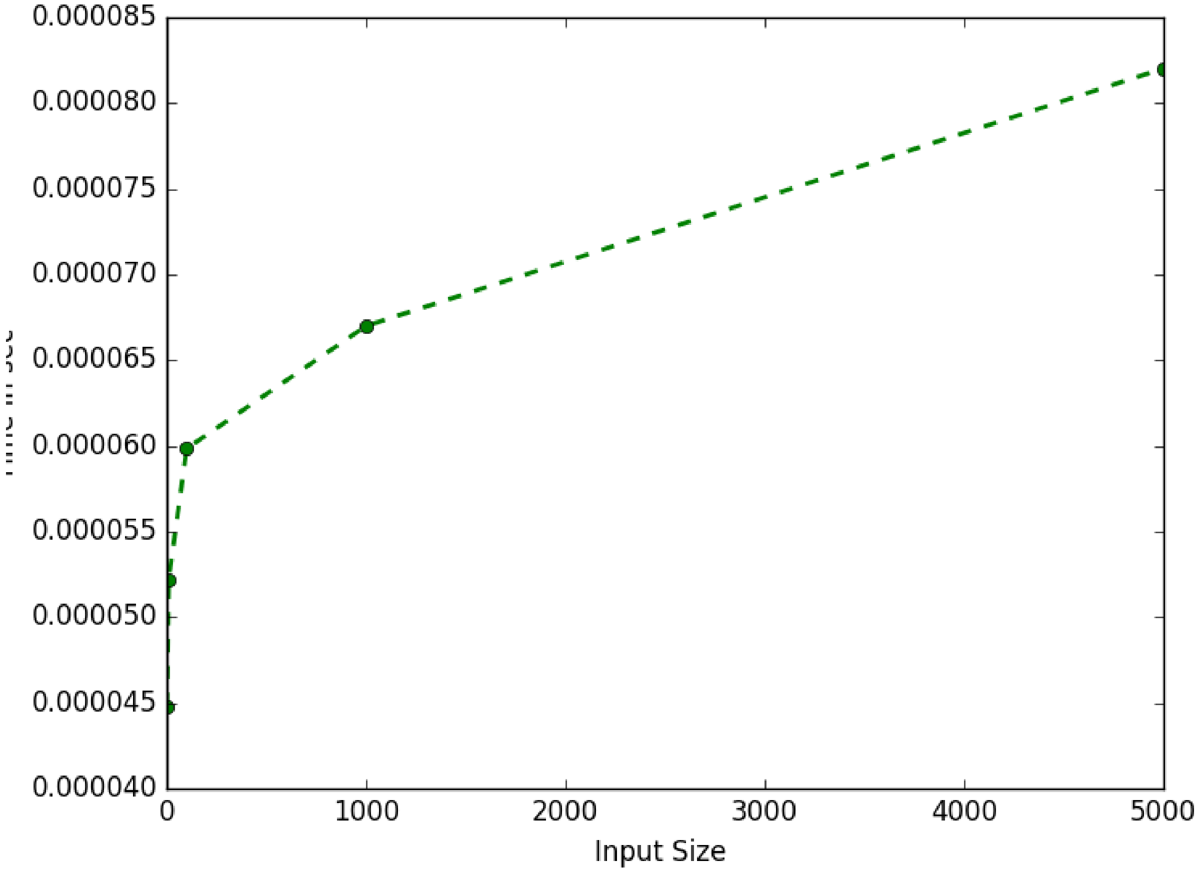
hence
T(n) = O(logn)

Input/Output:

Input is n = nth number of fibonacci to print.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
1	0.000002
10	0.000002
100	0.000009
1000	0.000027
5000	0.000141
10000	0.000257
50000	0.001142
100000	**0.002280**

Complexity Graph:



Conclusion :
In algorithm fibonacci number is calculated from formula generated by solving recurrence relation for fibonacci number, in that formula the only computational cost associated is of POWER function which is executing in log n time.

Q. 22 Own Problme K Flip binary search i.e. a list of sorted element is right shifted k time where 0<= k <=n where n is number of element in the list

Algorithm KFLIPBINARYSEARCH(a,l,r,x):

Input: x number to find in fliped list.
Output: result as if number is in the list or not.

```
IF (l<=r) THEN DO:
  set mid = l + (r-l)/2
  IF (a[mid]==x) THEN DO:
    RETURN mid
  ELSE IF (x>a[mid]) THEN DO:
    IF (x>a[mid] and x<=a[r]) THEN DO:
      RETURN KBINARYSEARCH(a,mid+1,r,x)
    ELSE DO:
      RETURN KBINARYSEARCH(a,l,mid-1,x)
  ELSE IF (x<a[mid]) THEN DO:
    IF (x<a[mid] and x>=a[l]) THEN DO:
      RETURN KBINARYSEARCH (a,l,mid-1,x)
    ELSE DO:
      RETURN KBINARYSEARCH (a,mid+1,r,x)
  ELSE DO:
    RETURN FALSE
```

Algorithm Analysis:

T(n) = T(n/2) + c ...... c for comparision associated cost.

hence
T(n) = O(logn)

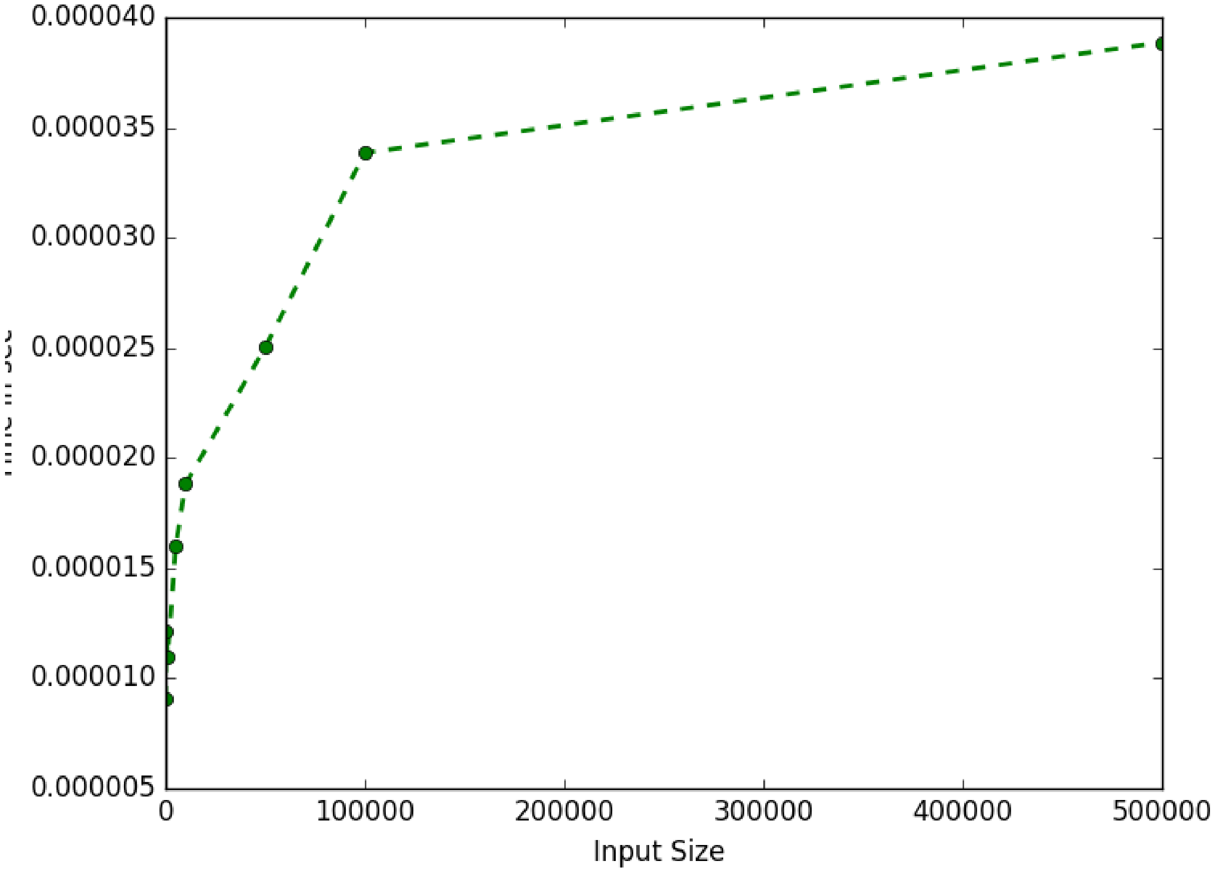
Input/Output:

Input is n = Size of list.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
10	9.05990600586e-06
100	1.21593475342e-05
1000	1.09672546387e-05
5000	1.59740447998e-05
10000	1.8835067749e-05
50000	2.50339508057e-05

100000	3.38554382324e-05
500000	3.88622283936e-05

Complexity Graph:



Conclusion :
It can be confirmed from graph that the actual complexity curve for implemented algorithm is similar to expected logn complexity.