Q. 24 Write a program for the following problem:
***INPUT:* A set S = { (xi; yi) 1 ≤ i ≤ n } of intervals over the real line.**
***OUTPUT:* A maximum cardinality subset S' of S such that no pair of intervals in S' overlap.**
Consider the following algorithm:
Repeat until S is empty
1. Select the interval I that overlaps the least number of other intervals.
2. Add I to final solution set S'.
3. Remove all intervals from S that overlap with I.
Prove or disprove that this algorithm solves the problem.

Algorithm MAXCARDINALITY(start,finish,S):

INPUT: A set S = { (xi; yi) 1 ≤ i ≤ n } of intervals over the real line.
OUTPUT: A maximum cardinality subset S' of S such that no pair of intervals in S' overlap.

```
  sort(finish);
  set n = len(s);
  set A = [Φ];
  set A = A U {(S[0])};
  set k = 1;
  FOR i 2 to n DO:
    IF start[i] >= finish[k]  THEN DO:
      set A = A U {(S[i])};
      set k = i;
  END LOOP
RETURN A;
```

Algorithm Analysis:

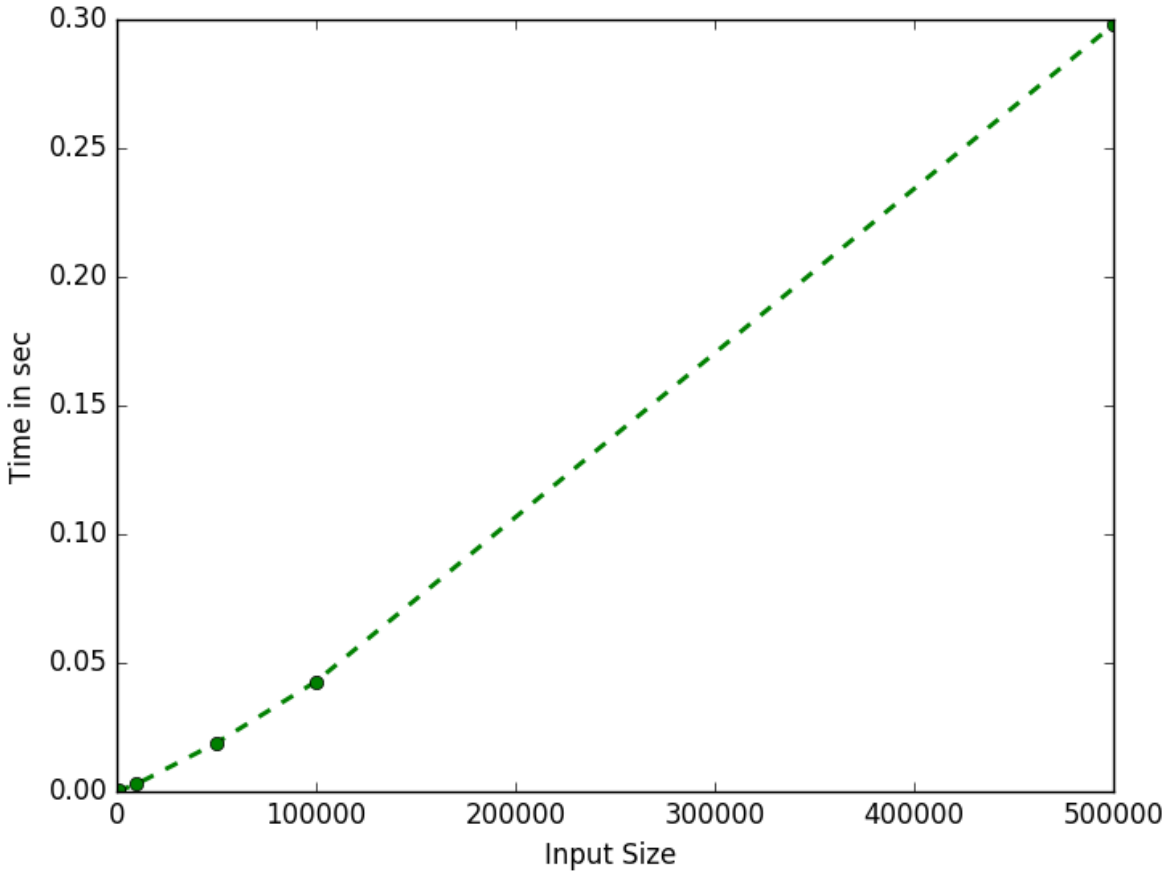
T(n) = O(nlogn)

Input/Output:

Input is n = number of input taken randomly.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
10	2.19345092773e-05
100	3.88622283936e-05
500	0.000200986862183
1000	0.000334024429321
10000	0.00289297103882
50000	0.0188808441162
100000	0.0427558422089
500000	**0.298007011414**

Complexity Graph:



Conclusion :

As expected curve show that it is non polynomial type. The above algorithm result in to solution because initialy we are selecting a set with least overlapping that means this set will affect other set least, hence removing sets with low number of overlap will ultimaltely remain with all set with non-overlapping elements and its cardinality should be max because in each iteration we are only removing elements with least number of overlap.

Q. 28 You wish to drive from point A to point B along a highway minimizing the time that you are stopped for gas. You are told beforehand the capacity C of you gas tank in liters, your rate F of fuel consumption in liters/kilometer, the rate r in liters/minute at which you can fill your tank at a gas station, and the locations A=x1,x2, ... , B=xn of the gas stations along the highway. So if you stop to fill your tank from 2 liters to 8 liters, you would have to stop for 6/r minutes.

Consider the following two algorithms:
(a) Stop at every gas station, and fill the tank with just enough gas to make it to the next gas station.
(b) Stop if and only if you don't have enough gas to make it to the next gas station, and if you stop, fill the tank up all the way.

For each algorithm write a program and also either prove or disprove that this algorithm correctly solves the problem. Your proof of correctness must use an exchange argument.

INPUT: station list, capacity of tank, mileage, rate of filling.
OUTPUT: time taken for journy.

Algorithm STOPATEVERYSTN(st,capacity,fuelMileage,rate):

```
set C = capacity;
set F = fuelMileage;
set R = rate;
set time = 0;
set old = 0;

FOR i 0 to |st| DO:
    set new = st[i];
    set km = st[i]-old;
    IF (C >= km/F ) THEN DO:
      set time = time + R*(km/F);
    ELSE DO:
       PRINT “Distance is more farther than it appear";
       BREAK;
    set old = st[i];

RETURN time;
```

Algorithm FULLATSTATION(st,capacity,fuelMileage,rate):

```
set C = capacity;
set F = fuelMileage;
set R = rate;
set time = R*C;
set old = float(0);
set fuelStatus = C;

FOR i 0 to |st| DO:
        set new = st[i];
        set km = st[i]-old;
        IF (fuelStatus < (km/F) ) THEN DO:
           set neededFuel = C-fuelStatus;
           set time = time + R*(neededFuel);
           set fuelStatus = C;
        set fuelStatus = fuelStatus - (km/F);
        old = st[i];
 RETURN time;
```

Conclusion :

In given both algorithm the first one is more optimum because at every station we are filling upto the capacity needed for the journy till next station hence we are already choosing minimum time(fuel) to be wasted making more greedy choice, on the other hand in second algorithm we are filling full tank even if we are not needing full tank. Hence in most of the case first algorithm will yeild better result and at most it will yeild result same as of second algorithm hence $1^{st} <= 2^{nd}$.

Q. 29 Write a program for the following problem:
Consider the following bridge crossing problem where n people with speeds s1; : : : ; sn wish to cross the bridge as quickly as possible. The rules remain:

- **It is nighttime and you only have one ash-light.**
- **A maximum of two people can cross at any one time**
- **Any party who crosses, either 1 or 2 people must have the ash-light with them.**
- **The ash-light must be walked back and forth, it cannot be thrown, etc.**
- **A pair must walk together at the rate of the slower person's pace.**

Give an efficient algorithm to find the fastest way to get a group of people across the bridge.
You must have a proof of correctness for your method.

INPUT: List of peoples with their walking speed.
OUTPUT: Fastest way for crossing the bridge.

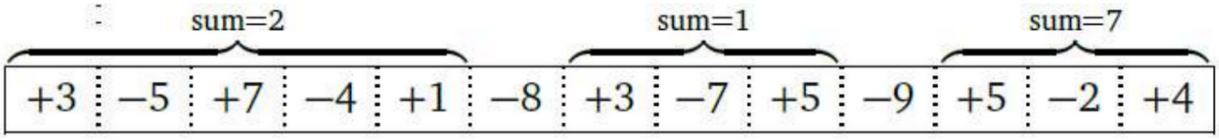
Algorithm ASHLIGHT(p):

Select Pair of fastest person and slowest person.
Cross bridge.
fastest person return.
take slowest in remaining.
again crosses the bridge.
repeat until all people crosses the bridge.

Conclusion :

Since minimum time required is sum of (time taken by all the persons + time take to return every time), since time taken by all persons is fixed, hence to minimise the time we need to minimise the time taken in return journy which could only be possible with taking fastest person for every return journy.

Q. 30 Suppose you are given an array A[1 .. n] of integers, each of which may be positive, negative, or zero. A contiguous subarray A[i .. j] is called a positive interval if the sum of its entries is greater than zero. Describe and analyze an algorithm to compute the minimum number of positive intervals that cover every positive entry in A. For example, given the following array as input, your algorithm should output the number 3.



INPUT: List of numbers.
OUTPUT: minimum number of positive intervals.

Algorithm FINDMININTER(a):

```
set sum = 0;
set m = 0;
set i = 0;
WHILE (i<n) DO:
  set sum = SUM(a,n-i,i-1);
  FOR j (n-1 to i-1) DO:
    IF sum > 0 THEN DO:
      set i =  j;
      set m = m + 1;
      BREAK;
    ELSE DO:
      set sum = sum - a[j] ;
  set i = i+1;

RETURN m;
```

Algorithm Analysis:

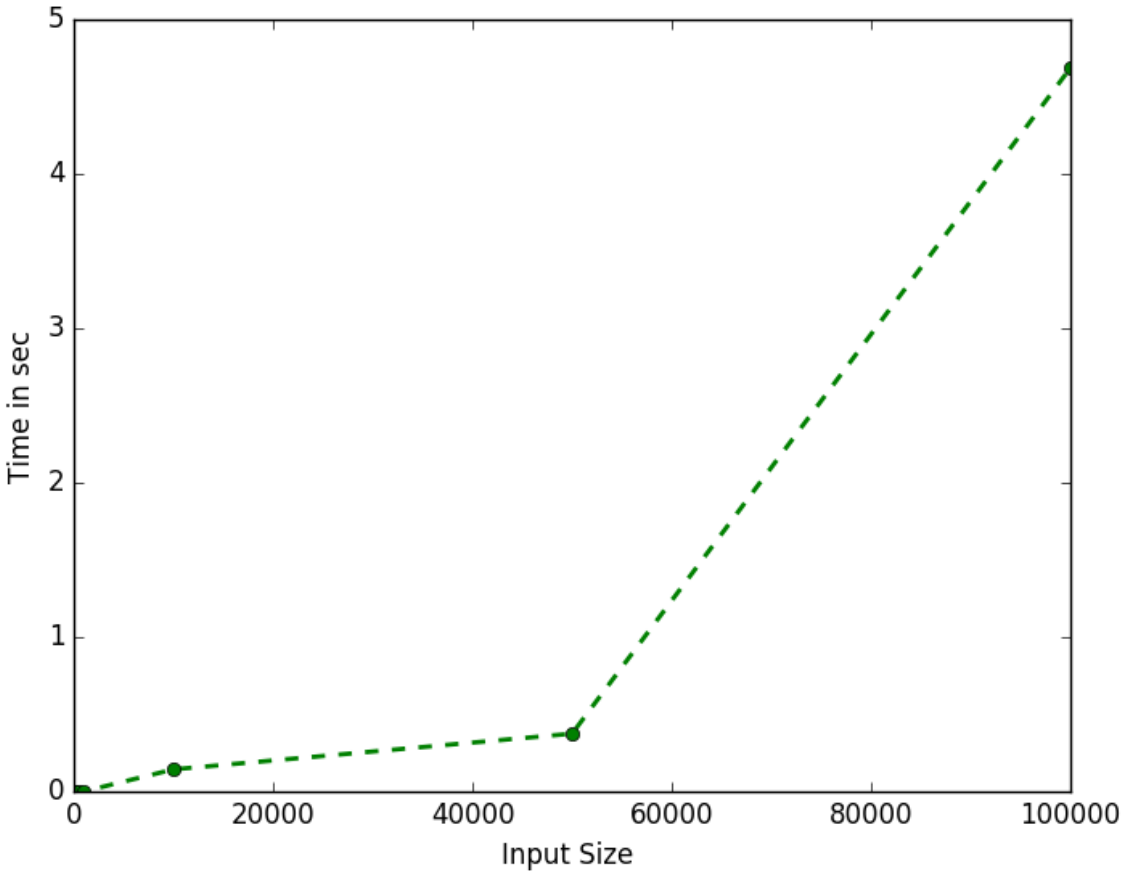
T(n) = $O(n^2)$

Input/Output:

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

Input	Time (in Sec)
10	5.3882598877e-05
100	4.31537628174e-05
500	8.98838043213e-05
1000	0.000221014022827
10000	0.143166065216
50000	**0.374448060989**
100000	**4.68612098694**

Complexity Graph:



Conclusion :

Looping for n time each time we are finding sum of list from 0 to x for xth iteration, after each iteration we remove element from list and repeat until sum till that element is Positive, thus increment counter by one and repeat till the end of list.